STLmc: Robust STL Model Checking of Hybrid Systems using SMT

Geunyeol Yu, Jia Lee, and Kyungmin Bae

Pohang University of Science and Technology, Pohang, Korea

Abstract. We present the STLmc model checker for signal temporal logic (STL) properties of hybrid systems. STLmc can perform STL model checking up to a robustness threshold $\epsilon > 0$ for a wide range of nonlinear hybrid systems with ordinary differential equations. Our tool utilizes the refutation-complete SMT-based model checking algorithm with various SMT solvers by reducing the robust STL model checking problem into the Boolean STL model checking problem. If STLmc does not find a counterexample, the system is guaranteed to be correct up to the given bounds and robustness threshold. We demonstrate the effectiveness of STLmc on a number of hybrid system benchmarks.

1 Introduction

Signal temporal logic (STL) [36] has emerged as a popular property specification formalism for hybrid systems. STL formulas describe linear-time properties of continuous real-valued signals. Because hybrid systems exhibit both discrete and continuous behaviors, STL provides a convenient and expressive way to specify important requirements of hybrid systems. STL has a vast range of applications on hybrid systems, including automotive systems [20,31], medical systems [41], robotics [28,46], IoT [9], smart cities [21,35], smart grids [45], etc.

Due to the infinite-state nature of hybrid systems with continuous dynamics, most techniques and tools for analyzing STL properties focus on monitoring and falsification. These techniques analyze concrete samples of signals obtained by simulating hybrid automata to monitor the system's behavior [15,17,30,37] or find counterexamples [1,4,42,48], often combined with stochastic optimization techniques [32]. To this end, STL monitoring and falsification use quantitative semantics that defines the *robustness degree* to indicate how well the formula is satisfied. However, these methods cannot be used to guarantee correctness.

Recently, several STL model checking techniques have been proposed for hybrid systems [6,34,40]. In particular, the SMT-based bounded model checking algorithms [6,34] are refutation-complete and thus can guarantee correctness up to bounds. These techniques are based on the Boolean semantics of STL instead of quantitative semantics. This is certainly a limitation for hybrid systems as small perturbations of signals can cause the system to violate the properties *verified* by Boolean STL model checking. Moreover, there is no tool with a convenient user interface that implements STL model checking techniques.

This paper presents the STLmc tool for robust STL model checking of hybrid systems. Our tool can verify that, up to bounds, the robustness degree of an STL formula φ is greater than a *robustness threshold* $\epsilon > 0$ for all possible behaviors of the system. We reduce the robust STL model checking problem to Boolean STL model checking using *ϵ-strengthening*, first proposed in [25] for first-order logic and extended to STL. We then apply the refutation-complete bounded model checking algorithm [6, 34] to build the SMT encoding of the resulting Boolean STL model checking problem, which can be solved using SMT solvers.

Apart from the robust STL model checking method, STLmc also implements several techniques to improve the usability and scalability of the tool:

- STLmc implements a generic interface to connect with various SMT solvers, such as Z3 [14], Yices2 [19], dReal [26]. Because dReal can (approximately) deal with nonlinear ordinary differential equations (ODEs), STLmc can also support hybrid systems with nonlinear ODE dynamics.
- STLmc implements parallelized two-step SMT solving to improve scalability. Instead of directly solving the complex encoding with ODEs, we first obtain a *discrete abstraction* without ODEs and find satisfying scenarios. We then check the *discrete refinements* of such scenarios using dReal in parallel.
- STLmc provides a visualization command to draw counterexample signals and robustness degrees. Such graphs intuitively explain why the robustness degree of the formula is greater than a given threshold, and thus greatly help in analyzing counterexamples and debugging hybrid systems.

We demonstrate the effectiveness of the STLmc tool on a number of hybrid system benchmarks—including linear, polynomial, and ODE dynamics—and nontrivial STL properties. The tool is available at `https://stlmc.github.io`.

The rest of this paper is organized as follows. Section 2 explains some background on robust STL model checking of hybrid systems using STLmc. Section 3 presents the STLmc tool and language. Section 4 presents the algorithm and implementation of STLmc. Section 5 shows the experimental results. Section 6 discusses the related work. Finally, Section 7 presents some concluding remarks.

2 Background

2.1 Hybrid Automata

Hybrid automata are widely used for formalizing cyber-physical systems (CPSs) that exhibit both discrete and continuous behaviors. In a hybrid automaton H, a set of *modes* Q specifies discrete states, and a finite set of real-valued *variables* $X = \{x_1, ..., x_l\}$ specifies continuous states. A state of H is a pair $\langle q, \vec{v}\rangle \in Q \times \mathbb{R}^l$ of mode q and real-valued vector $\vec{v}$. An initial condition $\mathit{init}(\langle q, \vec{v}\rangle)$ defines a set of initial states. An invariant condition $\mathit{inv}(\langle q, \vec{v}\rangle)$ defines a set of valid states. A flow condition $\mathit{flow}(\langle q, \vec{v}\rangle, t, \langle q, \vec{v}_t\rangle)$ defines a *continuous evolution* of variables X from a value $\vec{v}$ to $\vec{v}_t$ over duration t in mode q. A jump condition $\mathit{jump}(\langle q, \vec{v}\rangle, \langle q', \vec{v'}\rangle)$ defines a discrete transition from state $\langle q, \vec{v}\rangle$ to $\langle q', \vec{v'}\rangle$.

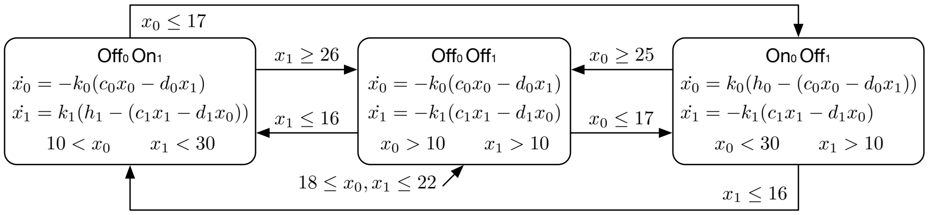
$x_0 \leq 17$
Off₀ On₁
$\dot{x_0} = -k_0(c_0x_0 - d_0x_1)$
$\dot{x_1} = k_1(h_1 - (c_1x_1 - d_1x_0))$
$10 < x_0$ $x_1 < 30$
$x_1 \geq 26$
$x_1 \leq 16$
Off₀ Off₁
$\dot{x_0} = -k_0(c_0x_0 - d_0x_1)$
$\dot{x_1} = -k_1(c_1x_1 - d_1x_0)$
$x_0 > 10$ $x_1 > 10$
$x_0 \geq 25$
$x_0 \leq 17$
On₀ Off₁
$\dot{x_0} = k_0(h_0 - (c_0x_0 - d_0x_1))$
$\dot{x_1} = -k_1(c_1x_1 - d_1x_0)$
$x_0 < 30$ $x_1 > 10$
$18 \leq x_0, x_1 \leq 22$
$x_1 \leq 16$

Fig. 1: A hybrid automaton for the networked thermostats.

Definition 1. *A hybrid automaton is a tuple* $H = (Q, X, \mathit{init}, \mathit{inv}, \mathit{jump}, \mathit{flow})$.

A *signal* σ represents a continuous execution of a hybrid automaton H, given by a function $[0, \tau) \to Q \times \mathbb{R}^l$ with a time bound $\tau > 0$. A signal σ is called *bounded* if $\tau < \infty$. A signal σ is called a *trajectory* of a hybrid automaton H, written $\sigma \in H$, if σ describes a valid behavior of H.

Definition 2. *For a hybrid automaton* H*, a signal* $\sigma : [0, \tau) \to Q \times \mathbb{R}^l$ *is a trajectory of* H*, written* $\sigma \in H$*, if there exist sequences of modes* $q_1, q_2, q_3, \ldots$ *and of times* $0 = t_0 < t_1 < \ldots < \tau$ *such that:*

- *the initial condition holds at time* t_0*: i.e.,* $\mathit{init}(\sigma(t_0))$ *holds;*
- *for* $i \geq 1$*, the values of* X *change from* $\sigma(t_{i-1})$ *for time* $t_i - t_{i-1}$ *by the flow condition, satisfying the invariant condition: i.e., for any* $t \in [t_{i-1}, t_i)$*:*

$$\mathit{flow}(\sigma(t_{i-1}), t - t_{i-1}, \sigma(t)) \quad \textit{and} \quad \mathit{inv}(\sigma(t))$$

- *for* $i \geq 1$*, a discrete jump happens at time* t_i*: i.e., for some* $s_i \in Q \times \mathbb{R}^l$*:*

$$\mathit{flow}(\sigma(t_{i-1}), t_i - t_{i-1}, s_i) \quad \textit{and} \quad \mathit{jump}(s_i, \sigma(t_i))$$

Example 1. There are two rooms connected by an open door. The temperature x_i of each room $i \in \{0, 1\}$ is controlled by each thermostat, depending on the heater's mode $q_i \in \{\mathsf{On}, \mathsf{Off}\}$ and the other room's temperature. The continuous dynamics of x_i can be given as ODEs as follows [5, 29]:

$$\dot{x_i} = \begin{cases} K_i(h_i - (c_ix_i - d_ix_{1-i})) & (\mathsf{On}) \\ -K_i(c_ix_i - d_ix_{1-i}) & (\mathsf{Off}), \end{cases}$$

where K_i, h_i, c_i, d_i are constants depending on the size of the room, the heater's power, and the size of the door. Figure 1 shows a hybrid automaton of our thermostat controllers. Initially, both heaters are off and the temperatures are between 18 and 22. The jumps between modes then define a control logic to keep the temperatures within a certain range using only one heater.

2.2 Signal Temporal Logic

Signal temporal logic (STL) is widely used to specify properties of hybrid systems [36]. The syntax of STL is defined by:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \, \mathbf{U}_I \, \varphi$$

where p denotes state propositions, and $I \subseteq \mathbb{R}_{\geq 0}$ is an interval of nonnegative real numbers. Examples of state propositions include relational expressions of the form $f(\vec{x}) \geq 0$ over variables X with a real-valued function $f : \mathbb{R}^l \to \mathbb{R}$. Other common boolean and temporal operators can be derived by equivalences:

$$\varphi \vee \varphi' \equiv \neg(\neg\varphi \wedge \neg\varphi'), \quad \Diamond_I \varphi \equiv \top \, \mathbf{U}_I \, \varphi,$$
$$\Box_I \varphi \equiv \neg\Diamond_I \neg\varphi, \quad \varphi \, \mathbf{R}_I \, \varphi' \equiv \neg((\neg\varphi) \, \mathbf{U}_I (\neg\varphi'))$$

Example 2. For Example 1, we consider the following STL formulas:

- $\Diamond_{[0,3]} \, (x_0 \geq 13 \; \mathbf{U}_{[0,\infty)} \; x_1 \leq 22)$
- $\Box_{[2,4]} \, (x_0 - x_1 \geq 4 \; \to \; \Diamond_{[3,10]} \, x_0 - x_1 \leq -3)$
- $\Box_{[0,10]} \, (x_0 > 23 \; \mathbf{R}_{[0,\infty)} \; x_0 - x_1 \geq 4)$

The Minkowski sum of two intervals I and J is denoted by $I + J$. E.g., $[a, b] + [c, d] = [a + c, b + d]$. For a singular interval $\{t\}$, the set $\{t\} + I$ is often written as $t + I$. We write $\sup_{a \in A} g(a)$ and $\inf_{a \in A} g(a)$ to denote the least upper bound and the greatest lower bound of the set $\{g(a) \mid a \in A\}$, respectively.

We consider a quantitative semantics of STL based on *robustness degrees* [17]. The semantics of a proposition p is specified as a function $p : Q \times \mathbb{R}^l \to \overline{\mathbb{R}}$, where $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, \infty\}$, that assigns to a state of H the degree to which p is true. For a state proposition of the form $f(\vec{x}) \geq 0$, the robustness degree is the value of $f(\vec{x})$ at a given state. E.g., the robustness degree of proposition $x_0 - x_1 \geq 4$ in Example 2 is the value of $x_0 - x_1 - 4$ at a given state.

The robustness degree of an STL formula can be defined as follows [17], where a time bound τ of a signal is explicitly taken into account.

Definition 3. *Given an STL formula φ, a signal $\sigma : [0, \tau) \to \mathbb{R}^l$, and a time $t \in [0, \tau)$, the robustness degree $\rho_\tau(\varphi, \sigma, t) \in \overline{\mathbb{R}}$ is defined inductively by:*

$$\begin{aligned}
\rho_\tau(p, \sigma, t) &= p(\sigma(t)) \\
\rho_\tau(\neg\varphi, \sigma, t) &= -\rho_\tau(\varphi, \sigma, t) \\
\rho_\tau(\varphi_1 \wedge \varphi_2, \sigma, t) &= \min(\rho_\tau(\varphi_1, \sigma, t), \rho_\tau(\varphi_2, \sigma, t)) \\
\rho_\tau(\varphi_1 \, \mathbf{U}_I \, \varphi_2, \sigma, t) &= \sup\nolimits_{t' \in (t+I) \cap [0,\tau)} \min(\rho_\tau(\varphi_2, \sigma, t'), \inf\nolimits_{t'' \in [t,t']} \rho_\tau(\varphi_1, \sigma, t''))
\end{aligned}$$

The robust STL model checking problem is to determine if the robustness degree of an STL formula φ is always greater than a given robustness threshold $\epsilon > 0$ for all possible trajectories of a hybrid automaton H.

Definition 4 (Robust STL Model Checking). *For a time bound $\tau > 0$, an STL formula φ is satisfied at time $t \in [0, \tau)$ on a hybrid automaton H with respect to a robustness threshold $\epsilon > 0$ iff for every trajectory $\sigma \in H$, $\rho_\tau(\varphi, \sigma, t) > \epsilon$.*

2.3 STL Boolean Semantic

We also introduce the Boolean semantics of STL, which was originally proposed as the semantics of STL in [36]. We consider the following definition from [34] that explicitly considers a time bound $\tau > 0$.

Definition 5. *Given an STL formula φ, a signal $\sigma : [0, \tau) \to \mathbb{R}^l$, and a time $t \in [0, \tau)$, the satisfaction of φ, denoted by $\sigma, t \models_\tau \varphi$ is inductively defined by:*

$$\begin{array}{ll} \sigma, t \models_\tau p & \textit{iff } t < \tau \textit{ and } p(\sigma(t)) = \top \\ \sigma, t \models_\tau \neg\varphi & \textit{iff } \sigma, t \not\models_\tau \varphi \\ \sigma, t \models_\tau \varphi_1 \wedge \varphi_2 & \textit{iff } \sigma, t \models_\tau \varphi_1 \textit{ and } \sigma, t \models_\tau \varphi_2 \\ \sigma, t \models_\tau \varphi_1 \, \mathbf{U}_I \, \varphi_2 & \textit{iff } \exists t' \in (t + I) \cap [0, \tau).\ \sigma, t' \models_\tau \varphi_2,\ \forall t'' \in [t, t'].\ \sigma, t'' \models_\tau \varphi_1 \end{array}$$

The following theorem shows the obvious relationship between the Boolean semantics and the quantitative semantics. If the robustness degree of φ is strictly positive (resp., negative), then φ is true (resp., false).

Theorem 1. *[17] For an STL formula φ, a signal σ, and a time $t \in [0, \tau)$, $\rho_\tau(\varphi, \sigma, t) > 0$ implies $\sigma, t \models_\tau \varphi$, and $\rho_\tau(\varphi, \sigma, t) < 0$ implies $\sigma, t \not\models_\tau \varphi$.*

3 The STLmc Tool

The STLmc tool can model check STL properties of hybrid automata, given three parameters $\epsilon > 0$ (robustness threshold), $\tau > 0$ (time bound), and $N \in \mathbb{N}$ (discrete bound). STLmc provides an expressive input format to easily specify a wide range of hybrid automata. STLmc also provides a visualization command to give an intuitive description of counterexamples.

3.1 Modeling language

The model input format of STLmc, inspired by dReach [33], consists of five sections: variable declarations, mode definitions, initial conditions, state propositions, and STL properties. Mode and continuous variables specify discrete and continuous states of hybrid automata. Mode definitions specify flow, jump, and invariant conditions. STL formulas can also include user-defined state propositions.

Variable Declarations. STLmc uses mode and continuous variables to specify discrete and continuous states of a hybrid automaton. *Mode variables* define a set of discrete modes Q, and *continuous variables* define a set of real-valued variables X.[1] We can also declare named constants whose values do not change.

Mode variables are declared with one of three types: `bool` variables with `true` and `false` values, `int` variables with integer values, and `real` variables with real values. E.g., the following declares a `bool` variable `b` and an `int` variable `i`:

[1] An assignment to mode and continuous variables corresponds to a single state.

```
bool b;    int i;
```

Continuous variables are declared with domain intervals. Domains can be any intervals of real numbers, including open, closed, and half-open intervals, E.g., the following declares two continuous variable `x` and `y`:

```
[0, 50] x;    (-1.1, 1) y;
```

Finally, constants are introduced with the `const` keyword. Constants can have Boolean, integer, or rational values. For example, the following declares a constant `k1` with a rational value 0.015:

```
const k1 = 0.015;
```

Mode Definitions. In STLmc, *mode blocks* define mode, jump, invariant, and flow conditions for a group of modes in a hybrid automaton. Multiple model blocks can be declared, and each mode block consists of four components:

```
{
    mode: ...
    inv:  ...
    flow: ...
    jump: ...
}
```

A `mode` component contains a semicolon-separated set of Boolean conditions over mode variables. The conjunction of these conditions represents a set of modes.[2] E.g., the following represents a set of two modes $\{(\mathit{true}, 1), (\mathit{true}, 2)\}$, provided there are two mode variables `b` (of `bool` type) and `i` (of `int` type):

```
b = true;    i > 0;    i < 3;
```

An `inv` component contains a semicolon-separated set of Boolean formulas over continuous variables. The conjunction of these conditions then represents the invariant condition for each mode in the mode block. For example, the following declares an invariant condition for two continuous variables `x` and `y`:

```
x < 30;    y > - 0.5;
```

A `flow` component contains either a system of ordinary differential equations (ODEs) or a closed-form solution of ODEs. In STLmc, a system of ODEs over continuous variables $x_1, \ldots, x_n$ is written as a semicolon-separated equation of the following form, where e_i denotes an expression over $x_1, \ldots, x_n$:

[2] We assume that the mode conditions of different mode blocks cannot be satisfied at the same time, which can be automatically detected by our tool.

```
d/dt[$x_1$] = $e_1(x_1, ..., x_n)$ ;
...
d/dt[$x_n$] = $e_n(x_1, ..., x_n)$ ;
```

A closed-form solution of ODEs is written as a set of continuous functions, parameterized by a time variable `t` and the initial values $x_1(0), ..., x_n(0)$:

```
$x_1$(t) = $e_1$(t, $x_1(0)$,...,$x_n(0)$) ;
...
$x_n$(t) = $e_n$(t, $x_1(0)$,...,$x_n(0)$) ;
```

A `jump` component contains a set of jump conditions *guard* `=>` *reset*, where *guard* and *reset* are Boolean conditions over mode and continuous variables. We use "primed" variables to denote states after jumps have occurred. E.g., the following defines a jump with four variables `b`, `i`, `x`, and `y` (declared above):

```
(and (i = 1) (x > 10)) => (and (b' = false) (i = 3) (x' = x) (y' = 0));
```

Initial Conditions. In the `init` section, an initial condition is declared as a set of Boolean formulas over mode and continuous variables. Similarly, the conjunction of these conditions represents a set of initial modes. E.g., the following shows an initial condition with variables `b`, `i`, `x`, and `y`:

```
init: (and (not b) (i = 0) (19.9 <= x) (y = 0));
```

STL Properties. In the `goal` section, STL properties are declared with labels. State propositions are arithmetic and relational expressions over mode and continuous variables, and labels can be omitted. For example, the following declares the first STL formula in Example 2 with label `f2`:

```
f2: [][2, 4]((x0 - x1 >= 4) -> <>[3, 10] (x0 - x1 <= -3));
```

To make it easy to write repeated propositions, "named" state propositions can be declared in the `proposition` section. For example, the above STL formula can be rewritten using two propositions `p1` and `p2` as follows:

```
proposition:
[p1]: x0 - x1 >= 4;
[p2]: x0 - x1 <= -3;

goal:
[f2]: [][2, 4](p1 -> <>[3, 10] p2);
```

Figure 2 shows the input model of the hybrid automaton described in the running of Example 1. Constants are introduced with the `const` keyword. Two mode variables `on0` and `on1` denote the heaters' modes. Continuous variables `x0`

```
const k0 = 0.015;       const k1 = 0.045;
const h0 = 100;         const h1 = 200;
const c0 = 0.98;        const c1 = 0.97;
const d0 = 0.01;        const d1 = 0.03;

int on0;                int on1;
[10, 35] x0;            [10, 35] x1;

{ mode: on0 = 0;         on1 = 1;
  inv:  10 < x0; x1 < 30;
  flow: d/dt[x0] = - k0 * (c0 * x0 - d0 * x1);
        d/dt[x1] = k1 * (h1 - (c1 * x1 - d1 * x0));
  jump: x0 <= 17 => (and (on0’ = 1) (on1’ = 0)
                         (x0’ = x0) (x1’ = x1));
        x1 >= 26 => (and (on1’ = 0) (on0’ = on0)
                         (x0’ = x0) (x1’ = x1));
}
{ mode: on0 = 1;         on1 = 0;
  inv:  x0 < 30; x1 > 10;
  flow: d/dt[x0] = k0 * (h0 - (c0 * x0 - d0 * x1));
        d/dt[x1] = - k1 * (c1 * x1 - d1 * x0);
  jump: x1 <= 16 => (and (on0’ = 0) (on1’ = 1)
                         (x0’ = x0) (x1’ = x1));
```

```
    x0 >= 25 => (and (on0’ = 0) (on1’ = on1)
                     (x0’ = x0) (x1’ = x1));
}
{ mode: on0 = 0;         on1 = 0;
  inv:  x0 > 10; x1 > 10;
  flow: d/dt[x0] = - k0 * (c0 * x0 - d0 * x1);
        d/dt[x1] = - k1 * (c1 * x1 - d1 * x0);
  jump:
    x0 <= 17 => (and (on0’ = 1) (on1’ = on1)
                     (x0’ = x0) (x1’ = x1));
    x1 <= 16 => (and (on1’ = 1) (on0’ = on0)
                     (x0’ = x0) (x1’ = x1));
}

init: on0 = 0;  18 <= x0;  x0 <= 22;
        on1 = 0;  18 <= x1;  x1 <= 22;

proposition:
  [p1]: x0 - x1 >= 4;    [p2]: x0 - x1 <= -3;

goal:
  [f1]: <>[0, 3](x0 >= 13 U[0, inf) x1 <= 22);
  [f2]: [][2, 4](p1 -> <>[3, 10] p2);
```

Fig. 2: An input model example

and `x1` are declared with domain intervals. There are three “mode blocks” that specify the three modes in Fig. 1 and their invariant, flow, jump conditions.

In mode blocks, a `mode` component includes a set of logic formulas over mode variables. An `inv` component contains a set of logic formulas over continuous variables. A `flow` component can include ODEs over continuous variables. A `jump` component contains a set of jump conditions *guard* `=>` *reset*, where *guard* and *reset* are logic formulas over mode and continuous variables, and “primed” variables denote states after the jump has occurred.

STL properties are declared in the `goal` section, and “named” propositions are declared in the `proposition` section. State propositions can include arithmetic and relational expressions over mode and continuous variables. For example, in Fig 2, the STL formula `f1` contains two state propositions $x_0 \geq 13$ and $x_1 \leq 22$, and the formula `f2` contains the user-defined propositions `p1` and `p2`.

3.2 Configuration

A *configuration file* is defined in a `.cfg` file. The configuration file specifies the analysis parameters for STLmc and underlying solvers. Analysis parameters for STLmc are listed in Table 1. Fig. 3 shows a configuration example.

Analysis parameters for STLmc (Table 1) are defined within curly braces introduced with the `common` keyword. There are two mandatory parameters and eight optional parameters. When mandatory parameters are not set, STLmc throws an exception.

Two parameters, `bound` and `time-bound`, are required parameters because the STLmc tool solves the robust STL model checking problem upto two bound

```
# STLmc configuration
common {
    # mandatory arguments
    # bound =
    # time-bound =

    threshold = 0.01            # positive rational number
    solver = auto               # dreal, yices, z3
    time-horizon = "time-bound"
    goal = "all"                # STL formula labels
    two-step = "false"          # on
    parallel = "false"          # on
    visualize = "false"         # on
    verbose = "false"           # print verbose messages
}

# underlying solver
z3      { logic = "QF_NRA" # QF_LRA }
yices   { logic = "QF_NRA" # QF_LRA }
dreal {
    precision = 0.001            # positive rational number
    ode-order = 5                # natural number
    ode-step = 0.001             # positive rational number
    executable-path = "../dreal" # dreal executable path
}
```

Fig. 3: A configuration example

parameters. A bound limits the number of mode changes and the number of *variable points*—at which the truth value of some STL subformula changes—in trajectories. A time bound bounds the time domain of trajectories. A threshold is a robustness threshold that bound the robustness degree of an STL formula. A time horizon T limits the maximum time duration of single modes in trajectories. If an STL formula is declared with a label, only that formula can be analyzed by providing the label as an argument to the `goal`. The parameters `two-step` and `parallel` enable the (parallelized) two-step optimization. When the `visualize` is set, extra data for visualization is generated.

We can choose different SMT solvers by setting an argument to the `solver`. The STLmc tool currently supports three SMT solvers: Z3 [14], Yices2 [19] and dReal [26]. The underlying solver is chosen depending on the flow conditions of a hybrid automaton. Z3 and Yices2 can deal with linear and polynomial functions, and dReal can deal with nonlinear ODEs—which is undecidable in general for hybrid automata—approximately up to a given precision $\delta > 0$.

Analysis parameters for each solver are defined within curly braces introduced with the *solver_name* keyword. For `z3` and `yices2` there is only one parameter, `logic`, which sets the background logic for SMT solving. There are four param-

Name	Explanation	Default
`bound`	a discrete bound $N \in \mathbb{N}$	-
`time-bound`	a time bound $\tau \in \mathbb{Q}^+$	-
`threshold`	a robustness threshold $\epsilon \in \mathbb{Q}^+$	0.01
`solver`	an SMT solver to be used (`z3`/`yices`/`dreal`)	`auto`
`time-horizon`	a mode duration bound	τ
`goal`	a list of STL goals to be analyzed	all
`two-step`	use the two-step optimization	disabled
`parallel`	parallelize the two-step optimization	disabled
`visualize`	generate extra visualization data	disabled
`verbose`	print more information of the execution	disabled

Table 1: STLmc configuration parameters.

eters for the dreal3 solver. A `precision` sets δ for the framework of δ-complete decision procedures. A `ode-order` sets an maximum order of taylor expansion of ordinary differential equations. A `ode-step` sets a time step for numerical calculation. A `executable-path` defines a path to the dreal executable file.

For flexibility in setting analysis parameters, three levels of configurations are provided: (1) default configuration, (2) model configuration, and (3) model specification configuration. The model specific configuration inherits the model configuration, and the model configuration inherits the default configuration.

The STLmc tool provides a predefined configuration file, named `default.cfg`, in the top directory. The default configuration sets all parameters to their default values except for mandatory arguments (Refer Fig. 3)

The model configuration specifies some analysis parameters related to a specific model. For example, we want to analyze the input model in Fig. 1 at bound 5 and time bound 30 with respect to robustness threshold $\epsilon = 0.1$. Then, we only need to change the following three parameters: bound, time-bound, and threshold. Figure 4 shows the model configuration that specifies these partial parameters. Other parameters such as goal, parallel, etc, inherit the values defined in `default.cfg`.

Furthermore, people may want to run a model using a different configuration for each formula. People can instantiate a model specific configurations to instantiate some analysis parameters in a model configuration. For the same example above, we want to analyze a formula `f2` with respect to $\epsilon = 2$. Then, we only need to change the following two parameters: goal and threshold. Figure 5 shows the model specific configuration that specifies these partial parameters. Other parameters such as bound, time-bound, etc, inherit the values defined in a model configuration.

```
# STLmc configuration
common { bound = 5
         time-bound = 30
         threshold = 0.1 }
```

Fig. 4: A model configuration example

```
# STLmc configuration
common { goal = f2
         threshold = 2 }
```

Fig. 5: A model specific configuration example

3.3 Command Line interface

The STLMC tool provides a command-line interface. The tool takes a model file, configuration files, and all parameters in the default configuration. The model file argument is required and other configuration arguments are optional.

```
$./stlmc [path to model file] \
         -defaul-cfg [path to default config file]\
         -model-cfg [path to model config file] \
         -model-specific-cfg [path to model specific config file] \
         -bound [int] ...
```

If the `-default-cfg` is not provided, the tool uses the `default.cfg` in the top directory. If the `-model-cfg` is not given, the tool looks for a configuration file *<model_file>.cfg* in the path to model file. [3] If some parameters in the configuration are given as command-line options, those parameters override the values specified in the configuration files.

Example 3. Consider the input model in Fig. 2 (`therm.model`), the input model configuration in Fig. 4 (`them-model.cfg`), and the input model specification configuration in Fig. 5 (`them-model-spec.cfg`). The following command found a counterexample of the formula `f2` at bound 5 with respect to robustness threshold $\epsilon = 2$ in 8 seconds using dReal.

```
$./stlmc ./therm.model -model-cfg therm-model.cfg  \
         -model-specific-cfg therm-model-spec.cfg \
         -solver dreal -two-step -parallel -visualize
goal: []_[2.0,4.0] (p1 -> (<>_[3.0,10.0] p2))
result: counterexample found (bound 2)
running time: 7.46335 seconds
```

[3] If there is no matched model configuration file, the model configuration inherits the default configuration.

The following command verifies the formula f1 up to bounds $N = 5$ and $\tau = 30$ with respect to robustness threshold $\epsilon = 1$ in 817 seconds using dReal.

```
$./stlmc ./therm.model -bound 5 -time-bound 30 -threshold 1 \
        -goal f1 -solver dreal -two-step -parallel
goal : (<>_[0.0,3.0] ((x0 >= 13) U_[0.0,inf) (x1 <= 22)))
result : True
running time 816.99192 seconds
```

3.4 Visualization

The STLmc tool provides a script to visualize counterexamples for robust STL model checking. The visualization script takes a counterexample file and a visualization configuration file and returns graphs representing counterexample trajectories and robustness degrees.

```
./stlmc-vis [path to counterexample file] \
          -cfg [path to configuration file]
```

The counterexample file is specified in a .counterexample file. The file is created when there is a counterexample and the visualize option is set. The configuration file contains following things: (1) continuous variables, (2) STL subformula labels, (3) output format, and (4) group of variables. The file contains variable information of continuous variables and STL subformulas. The STLmc tool supports "html" and "pdf" as output formats. The states of variables in a group are plotted on one graph. Only variables of the same type can be grouped together.[4] Variables not assigned to a group are grouped together according to their type. Figure 6 shows a visualization configuration file of our thermostat controllers.

```
{   # continuous variables: x0, x1
    # STL subformula labels:
    # f2 --> [][2, 4]((x0 - x1 >= 4) -> <>[3,10](x0 - x1 <= -3))
    # f2_1 --> (x0 - x1 >= 4) -> <>[3,10](x0 - x1 <= -3)
    # f2_2 --> not (x0 - x1 >= 4)
    # f2_3 --> <>[3,10](x0 - x1 <= -3)
    # p_1 --> x0 - x1 >= 4
    # p_2 --> x0 - x1 <= -4

    # output = pdf # html
    group { (x0, x1), (f2, f2_1) (f2_2, f2_3) (p_1, p_2) }
```

Fig. 6: A visualization configuration example

[4] For example, if there is a group $(x0, f2)$, the tool will throw an error because the types of $x0$ and $f2$ are real and bool, respectively.

Example 4. Consider the conunterexample file for `f2` in Example 3 (`therm.model`) and the visualization configuration file in Fig. 6 (`therm_vis.cfg`). The following command generate a PDF image. in Fig. 7.

```
./stlmc-vis therm.counterexample -cfg therm_vis.cfg
```

The robustness degree of `f2` is less than ϵ at time 0, since the robustness degree of $\texttt{f2}_1$ goes below ϵ in the interval $[2, 4]$, which is because both the degrees of $\texttt{f2}_2$ and $\texttt{f2}_3$ are less than ϵ in $[2, 4]$. The robustness degree of $\texttt{f2}_3$ is less than ϵ in $[2, 4]$, since the robustness degree of $\texttt{p}_2$ is less than ϵ in $[5, 14] = [2, 4] + [3, 10]$.

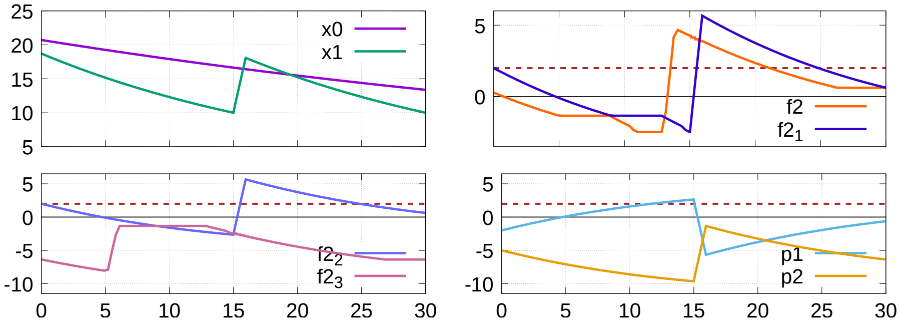

Fig. 7: Visualization of a counterexample (horizontal dotted lines denote $\epsilon = 2$).

4 Algorithms and Implementation

This section explains the algorithms and implementation of our STLmc tool. Section 4.1 explains how the robust STL model checking problem can be reduced into the Boolean STL model checking problem. Section 4.2 summarizes the Boolean STL model checking algorithm [6,34]. Section 4.3 explains a two-step solving optimization to improve the performance for ODEs. Finally, Section 4.4 presents the tool's architecture and implementation.

4.1 Reduction to Boolean STL Model Checking

As usual for model checking, robust STL model checking is equivalent to finding a counterexample. We can easily see that an STL formula φ is not satisfied in a hybrid automata H with respect to a robustness threshold $\epsilon > 0$ iff there is a trajectory that the robustness degree of $\neg\varphi$ is greater than or equal to $-\epsilon$.

Corollary 1. *For a hybrid automaton H, a time $t \in [0, \tau)$, and a robustness threshold $\epsilon > 0$, an STL formula φ is not satisfied at t in H with respect to ϵ iff there exists a trajectory $\sigma \in H$ such that $\rho_\tau(\neg\varphi, \sigma, t) \geq -\epsilon$.*

Proof. By definition, φ is satisfied at t in H with respect to ϵ iff for any trajectory $\sigma \in H$, $\rho_\tau(\varphi, \sigma, t) > \epsilon$ holds. Notice that $\neg(\forall \sigma \in H.\ \rho_\tau(\varphi, \sigma, t) > \epsilon)$ iff $\exists \sigma \in H.\ \rho_\tau(\varphi, \sigma, t) \leq \epsilon$. The formula is equal to $\exists \sigma \in H.\ -\rho_\tau(\varphi, \sigma, t) \geq -\epsilon$. Thus, $\exists \sigma \in H.\ \rho_\tau(\neg\varphi, \sigma, t) \geq -\epsilon$ by definition. □

Our goal is to reduce finding a counterexample of robust STL model checking into finding a counterexample of Boolean STL model checking. Let us first a simple case, say, a state proposition $x \geq 0$. Its robust model checking with respect ϵ is equivalent to finding a counterexample $\sigma \in H$ with $\rho_\tau(-x > 0, \sigma, t) \geq -\epsilon$ by Corollary 1, which is equivalent to $\rho_\tau(-x > -\epsilon, \sigma, t) \geq 0$. Notice that $-x > -\epsilon$ can be obtained by *weakening* $-x \geq 0$ by ϵ.

The notions of ϵ-weakening and ϵ-strengthening are first introduced in [25] for first-order formulas. In this paper, we extend the definitions of ϵ-weakening and ϵ-strengthening to STL formulas as follows.

Definition 6. *The ϵ-weakening $\varphi^{-\epsilon}$ and ϵ-strengthening $\varphi^{+\epsilon}$ of φ are defined as follows: $(p^{-\epsilon})(s) = p(s) - \epsilon$ and $(p^{+\epsilon})(s) = p(s) + \epsilon$ for a state s, and:*

$$\begin{array}{lll} (\neg\varphi)^{-\epsilon} \equiv \neg(\varphi^{+\epsilon}) & (\varphi_1 \wedge \varphi_2)^{-\epsilon} \equiv \varphi_1^{-\epsilon} \wedge \varphi_2^{-\epsilon} & (\varphi_1 \, \mathbf{U}_I \, \varphi_2)^{-\epsilon} \equiv \varphi_1^{-\epsilon} \, \mathbf{U}_I \, \varphi_2^{-\epsilon} \\ (\neg\varphi)^{+\epsilon} \equiv \neg(\varphi^{-\epsilon}) & (\varphi_1 \wedge \varphi_2)^{+\epsilon} \equiv \varphi_1^{+\epsilon} \wedge \varphi_2^{+\epsilon} & (\varphi_1 \, \mathbf{U}_I \, \varphi_2)^{+\epsilon} \equiv \varphi_1^{+\epsilon} \, \mathbf{U}_I \, \varphi_2^{+\epsilon} \end{array}$$

The following lemmas state that relationships between the quantitative semantics of φ and the Boolean semantics of $\varphi^{+\epsilon}$ and $\varphi^{-\epsilon}$.

Lemma 1. *For a signal σ and an STL property φ:*

$$\sigma, t \not\models_\tau \varphi^{+\epsilon} \ \Rightarrow\ \rho_\tau(\varphi, \sigma, t) \leq \epsilon \quad \textit{and} \quad \sigma, t \models_\tau \varphi^{-\epsilon} \ \Rightarrow\ \rho_\tau(\varphi, \sigma, t) \geq -\epsilon$$

Proof. The proof is by structural induction on φ.

For $\varphi = p$, if $\sigma, t \not\models_\tau p^{+\epsilon}$, then $p^{+\epsilon}(\sigma(t)) = \bot$ by definition. If $\sigma, t \models_\tau p^{-\epsilon}$, then $p^{-\epsilon}(\sigma(t)) = \top$ by definition. There are two possible propositions forms, $f(\vec{x}) \geq 0$ *and* $f(\vec{x}) > 0$. For $p = f(\vec{x}) \geq 0$, ϵ-strengthening $p^{+\epsilon}$ of p is $f(\vec{x}) \geq \epsilon$ and ϵ-weakening $p^{-\epsilon}$ of p is $f(\vec{x}) \geq -\epsilon$ by definition.

Suppose $\sigma, t \not\models_\tau p^{+\epsilon}$. Then, $f(\sigma(t)) < \epsilon$ is satisfied. The robustness degree $\rho_\tau(f(\vec{x}) \geq 0, \sigma, t)$ is $f(\sigma(t))$ by definition. Therefore, $\rho_\tau(f(\vec{x}) \geq 0, \sigma, t) < \epsilon$. If $\rho_\tau(f(\vec{x}) \geq 0, \sigma, t) < \epsilon$, then $\rho_\tau(f(\vec{x}) \geq 0, \sigma, t) \leq \epsilon$.

Suppose $\sigma, t \models_\tau p^{-\epsilon}$. Then, $f(\sigma(t)) \geq -\epsilon$ is satisfied. The robustness degree $\rho_\tau(f(\vec{x}) \geq 0, \sigma, t)$ is $f(\sigma(t))$ by definition. Therefore, $\rho_\tau(f(\vec{x}) \geq 0, \sigma, t) \geq -\epsilon$.

For $p = f(\vec{x}) > 0$, ϵ-strengthening $p^{+\epsilon}$ of p is $f(\vec{x}) > \epsilon$ and ϵ-weakening $p^{-\epsilon}$ of p is $f(\vec{x}) > -\epsilon$ by definition. Suppose $\sigma, t \not\models_\tau p^{+\epsilon}$. Then, $f(\sigma(t)) \leq \epsilon$ is satisfied. Since $\rho_\tau(f(\vec{x}) > 0, \sigma, t) = f(\sigma(t))$, $\rho_\tau(f(\vec{x}) > 0, \sigma, t) \leq \epsilon$.

Suppose $\sigma, t \models_\tau p^{-\epsilon}$. Then, $f(\sigma(t)) > -\epsilon$ is satisfied. Thus, $\rho_\tau(f(\vec{x}) > 0, \sigma, t) > -\epsilon$, since $\rho_\tau(f(\vec{x}) > 0, \sigma, t) = f(\sigma(t))$ by definition. The ϵ-weakening of p is $f(\vec{x}) > -\epsilon$ by definition. Thus, $\rho_\tau(f(\vec{x}) > 0, \sigma, t) \geq -\epsilon$.

For $\varphi = \neg\phi$, $\sigma, t \not\models_\tau (\neg\phi)^{+\epsilon}$ iff $\sigma, t \not\models_\tau \neg(\phi^{-\epsilon})$, since $(\neg\phi)^{+\epsilon} \equiv \neg(\phi^{-\epsilon})$ by definition. Thus, $\sigma, t \not\models_\tau \neg(\phi^{-\epsilon})$ iff $\sigma, t \models_\tau (\phi^{-\epsilon})$.

If $\sigma, t \models_\tau (\phi^{-\epsilon})$, then $\rho_\tau(\phi, \sigma, t) \geq -\epsilon$ by induction hypothesis. By definition, $\rho_\tau(\phi, \sigma, t) \geq -\epsilon$ iff $-\rho_\tau(\phi, \sigma, t) \leq \epsilon$. Thus, $\rho_\tau(\neg\phi, \sigma, t) \leq \epsilon$, since $-\rho_\tau(\phi, \sigma, t) = \rho_\tau(\neg\phi, \sigma, t)$ by definition.

Suppose $\sigma, t \models_\tau (\neg\phi)^{-\epsilon}$. Then, $\sigma, t \models_\tau \neg(\phi^{+\epsilon})$ is also satisfied, since $(\neg\phi)^{-\epsilon} \equiv \neg(\phi^{+\epsilon})$ by definition. Thus, $\sigma, t \models_\tau \neg(\phi^{+\epsilon})$ iff $\sigma, t \not\models_\tau (\phi^{+\epsilon})$.

If $\sigma, t \not\models_\tau (\phi^{+\epsilon})$, then $\rho_\tau(\phi, \sigma, t) \leq \epsilon$ by induction hypothesis. By definition, $\rho_\tau(\phi, \sigma, t) \leq \epsilon$ iff $-\rho_\tau(\phi, \sigma, t) \geq -\epsilon$. Thus, $\rho_\tau(\neg\phi, \sigma, t) \geq -\epsilon$, since $-\rho_\tau(\phi, \sigma, t) = \rho_\tau(\neg\phi, \sigma, t)$ by definition.

For $\varphi = \varphi_1 \wedge \varphi_2$, $\sigma, t \not\models_\tau (\varphi_1 \wedge \varphi_2)^{+\epsilon}$ iff $\sigma, t \not\models_\tau \varphi_1^{+\epsilon} \vee \sigma, t \not\models_\tau \varphi_2^{+\epsilon}$ by definition. by induction hypothesis Then, by induction hypothesis,

$$\rho_\tau(\varphi_1, \sigma, t) \leq \epsilon \,\vee\, \rho_\tau(\varphi_2, \sigma, t) \leq \epsilon.$$

Thus, $\rho_\tau(\varphi_1 \wedge \varphi_2, \sigma, t) \leq \epsilon$, since $\rho_\tau(\varphi_1 \wedge \varphi_2, \sigma, t) = \min(\rho_\tau(\varphi_1, \sigma, t), \rho_\tau(\varphi_2, \sigma, t))$ and $\rho_\tau(\varphi_1, \sigma, t) \leq \epsilon \,\vee\, \rho_\tau(\varphi_2, \sigma, t) \leq \epsilon$.

Suppose $\sigma, t \models_\tau (\varphi_1 \wedge \varphi_2)^{-\epsilon}$. Then, $\sigma, t \models_\tau \varphi_1^{-\epsilon} \wedge \sigma, t \models_\tau \varphi_2^{-\epsilon}$ by definition. Thus, by induction hypothesis, $\rho_\tau(\varphi_1, \sigma, t) \geq -\epsilon \,\wedge\, \rho_\tau(\varphi_2, \sigma, t) \geq -\epsilon$. Therefore, $\rho_\tau(\varphi_1 \wedge \varphi_2, \sigma, t) \geq -\epsilon$, since $\rho_\tau(\varphi_1 \wedge \varphi_2, \sigma, t) = \min(\rho_\tau(\varphi_1, \sigma, t), \rho_\tau(\varphi_2, \sigma, t))$ and $\rho_\tau(\varphi_1, \sigma, t) \geq -\epsilon \,\wedge\, \rho_\tau(\varphi_2, \sigma, t) \geq -\epsilon$.

For $\varphi = \varphi_1 \,\mathbf{U}_I\, \varphi_2$, suppose $\sigma, t \not\models_\tau (\varphi_1 \,\mathbf{U}_I\, \varphi_2)^{+\epsilon}$. Then, for all time points $t' \in (t + I) \cap [0, \tau)$, $\sigma, t' \not\models_\tau \varphi_2^{+\epsilon} \vee\; \exists t'' \in [t, t'], \sigma, t'' \not\models_\tau \varphi_1^{+\epsilon}$ by definition. Therefore, by induction hypothesis,

$$\forall t' \in (t + I) \,\cap\, [0, \tau).(\rho_\tau(\varphi_2, \sigma, t') \leq \epsilon \,\vee\, \exists t'' \in [t, t'], \rho_\tau(\varphi_1, \sigma, t'') \leq \epsilon)$$

If $\exists t'' \in [t, t'], \rho_\tau(\varphi_1, \sigma, t'') \leq \epsilon$, then $\inf_{t'' \in [t,t']} \rho_\tau(\varphi_1, \sigma, t'') \leq \epsilon$. In summary,

$$\forall t' \in (t + I) \,\cap\, [0, \tau), (\rho_\tau(\varphi_2, \sigma, t') \leq \epsilon \,\vee\, \inf_{t'' \in [t,t']} \rho_\tau(\varphi_1, \sigma, t'') \leq \epsilon).$$

Since $\forall t' \in t + I \,\cap\, [0, \tau), (\min(\rho_\tau(\varphi_2, \sigma, t'), \inf_{t'' \in [t,t']} \rho_\tau(\varphi_1, \sigma, t'')) \leq \epsilon)$

$$\sup_{t' \in t + I \cap [0,\tau)} \min(\rho_\tau(\varphi_2, \sigma, t'), \inf_{t'' \in [t,t']} \rho_\tau(\varphi_1, \sigma, t'')) \leq \epsilon.$$

Therefore, $\rho_\tau(\varphi, \sigma, t) \leq \epsilon$ by definition.

Suppose $\sigma, t \models_\tau (\varphi_1 \,\mathbf{U}_I\, \varphi_2)^{-\epsilon}$. Then, there is a time point $t' \in (t + I) \cap [0, \tau)$ such that $\sigma, t' \models_\tau \varphi_2^{-\epsilon} \wedge\; \forall t'' \in [t, t'], \sigma, t'' \models_\tau \varphi_1^{-\epsilon}$ by definition. Then, $\rho_\tau(\varphi_2, \sigma, t') \geq -\epsilon \,\wedge\, \forall t'' \in [t, t'], \rho_\tau(\varphi_1, \sigma, t'') \geq -\epsilon$ by induction hypothesis. If $\forall t'' \in [t, t'], \rho_\tau(\varphi_1, \sigma, t'') \geq -\epsilon$, then $\inf_{t'' \in [t,t']} \rho_\tau(\varphi_1, \sigma, t'') \geq -\epsilon$. In summary,

$$\exists t' \in (t + I) \,\cap\, [0, \tau), (\rho_\tau(\varphi_2, \sigma, t') \geq -\epsilon \,\wedge\, \inf_{t'' \in [t,t']} \rho_\tau(\varphi_1, \sigma, t'') \geq -\epsilon).$$

Since $\exists t' \in t + I \,\cap\, [0, \tau), \min(\rho_\tau(\varphi_2, \sigma, t'), \inf_{t'' \in [t,t']} \rho_\tau(\varphi_1, \sigma, t'')) \geq -\epsilon$,

$$\sup_{t' \in t + I \cap [0,\tau)} \min(\rho_\tau(\varphi_2, \sigma, t'), \inf_{t'' \in [t,t']} \rho_\tau(\varphi_1, \sigma, t'')) \geq -\epsilon.$$

Therefore, $\rho_\tau(\varphi, \sigma, t) \geq -\epsilon$ by definition. □

Lemma 2. *For a signal σ and an STL property φ:*

$$\sigma, t \models_\tau \varphi^{+\epsilon} \Rightarrow \rho_\tau(\varphi, \sigma, t) \geq \epsilon \quad \textit{and} \quad \sigma, t \not\models_\tau \varphi^{-\epsilon} \Rightarrow \rho_\tau(\varphi, \sigma, t) \leq -\epsilon$$

Proof. The proof is by structural induction on φ.

For $\varphi = p$, if $\sigma, t \models_\tau p^{+\epsilon}$, then $p^{+\epsilon}(\sigma(t)) = \top$ by definition. If $\sigma, t \not\models_\tau p^{-\epsilon}$, then $p^{-\epsilon}(\sigma(t)) = \bot$ by definition. There are two possible state propositions forms, $f(\vec{x}) \geq 0$ and $f(\vec{x}) > 0$. For $p = f(\vec{x}) \geq 0$, ϵ-strengthening $p^{+\epsilon}$ of p is $f(\vec{x}) \geq \epsilon$ and ϵ-weakening $p^{-\epsilon}$ of p is $f(\vec{x}) \geq -\epsilon$ by definition.

Suppose $\sigma, t \models_\tau p^{+\epsilon}$. Then, $f(\sigma(t)) \geq \epsilon$ is satisfied. The robustness degree $\rho_\tau(f(\vec{x}) \geq 0, \sigma, t)$ is $f(\sigma(t))$ by definition. Therefore, $\rho_\tau(f(\vec{x}) \geq 0, \sigma, t) \geq \epsilon$.

Suppose $\sigma, t \not\models_\tau p^{-\epsilon}$. Then, $f(\sigma(t)) < -\epsilon$ is satisfied. The robustness degree $\rho_\tau(f(\vec{x}) \geq 0, \sigma, t)$ is $f(\sigma(t))$ by definition. Therefore, $\rho_\tau(f(\vec{x}) \geq 0, \sigma, t) < -\epsilon$.

If $\rho_\tau(f(\vec{x}) \geq 0, \sigma, t) < -\epsilon$, then $\rho_\tau(f(\vec{x}) \geq 0, \sigma, t) \leq -\epsilon$.

For $p = f(\vec{x}) > 0$, ϵ-strengthening $p^{+\epsilon}$ of p is $f(\vec{x}) > \epsilon$ and ϵ-weakening $p^{-\epsilon}$ of p is $f(\vec{x}) > -\epsilon$ by definition. Suppose $\sigma, t \models_\tau p^{+\epsilon}$. Then, $f(\sigma(t)) > \epsilon$ is satisfied. Thus, $\rho_\tau(f(\vec{x}) > 0, \sigma, t) > \epsilon$, since $\rho_\tau(f(\vec{x}) > 0, \sigma, t) = f(\sigma(t))$ by definition. If $\rho_\tau(f(\vec{x}) > 0, \sigma, t) > \epsilon$, then $\rho_\tau(f(\vec{x}) > 0, \sigma, t) \geq \epsilon$.

Suppose $\sigma, t \not\models_\tau p^{-\epsilon}$. Then, $f(\sigma(t)) \leq -\epsilon$ is satisfied. Thus, $\rho_\tau(f(\vec{x}) > 0, \sigma, t) \leq -\epsilon$, since $\rho_\tau(f(\vec{x}) > 0, \sigma, t) = f(\sigma(t))$.

For $\varphi = \neg\phi$, suppose $\sigma, t \models_\tau (\neg\phi)^{+\epsilon}$. Then, $\sigma, t \not\models_\tau \phi^{-\epsilon}$ by definition, since $(\neg\varphi)^{+\epsilon} \equiv \neg(\varphi^{-\epsilon})$. If $\sigma, t \not\models_\tau \phi^{-\epsilon}$ is satisfied, then $\rho_\tau(\phi, \sigma, t) \leq -\epsilon$ by induction hypothesis. Thus, $\rho_\tau(\phi, \sigma, t) \leq -\epsilon$ iff $\rho_\tau(\neg\phi, \sigma, t) \geq \epsilon$, since $\rho_\tau(\neg\phi, \sigma, t) = -\rho_\tau(\phi, \sigma, t)$ by definition.

Suppose $\sigma, t \not\models_\tau (\neg\phi)^{-\epsilon}$. Then, $\sigma, t \models_\tau \phi^{+\epsilon}$, since $(\neg\phi)^{-\epsilon} \equiv \neg(\phi^{+\epsilon})$ by definition. If $\sigma, t \models_\tau \phi^{+\epsilon}$ is satisfied, then $\rho_\tau(\phi, \sigma, t) \geq \epsilon$ by induction hypothesis. $\rho_\tau(\phi, \sigma, t) \geq \epsilon$ iff $\rho_\tau(\neg\phi, \sigma, t) \leq -\epsilon$, since $\rho_\tau(\neg\phi, \sigma, t) = -\rho_\tau(\phi, \sigma, t)$.

For $\varphi = \varphi_1 \wedge \varphi_2$, suppose $\sigma, t \models_\tau (\varphi_1 \wedge \varphi_2)^{+\epsilon}$. Then, $\sigma, t \models_\tau \varphi_1^{+\epsilon} \wedge \sigma, t \models_\tau \varphi_1^{+\epsilon}$ by definition. Thus, by induction hypothesis,

$$\rho_\tau(\varphi_1, \sigma, t) \geq \epsilon \wedge \rho_\tau(\varphi_2, \sigma, t) \geq \epsilon.$$

Therefore, $\rho_\tau(\varphi_1 \wedge \varphi_2) = \min(\rho_\tau(\varphi_1, \sigma, t), \rho_\tau(\varphi_2, \sigma, t)) \geq \epsilon$.

Suppose $\sigma, t \not\models_\tau (\varphi_1 \wedge \varphi_2)^{-\epsilon}$. Then, $\sigma, t \not\models_\tau \varphi_1^{-\epsilon} \vee \sigma, t \not\models_\tau \varphi_2^{-\epsilon}$ by definition. Thus, $\rho_\tau(\varphi_1, \sigma, t) \leq -\epsilon \vee \rho_\tau(\varphi_2, \sigma, t) \leq -\epsilon$ by induction hypothesis. Then, $\min(\rho_\tau(\varphi_1, \sigma, t), \rho_\tau(\varphi_2, \sigma, t)) \leq -\epsilon$. Therefore, $\rho_\tau(\varphi_1 \wedge \varphi_2) \leq -\epsilon$.

For $\varphi = \varphi_1 \, \mathbf{U}_I \, \varphi_2$, suppose $\sigma, t \models_\tau (\varphi_1 \, \mathbf{U}_I \, \varphi_2)^{+\epsilon}$. Then, there is a time point $t' \in (t + I) \cap [0, \tau)$ such that $\sigma, t' \models_\tau \varphi_2^{+\epsilon} \wedge \forall t'' \in [t, t'], \sigma, t'' \models_\tau \varphi_1^{+\epsilon}$ by definition. Therefore, by induction hypothesis,

$$\rho_\tau(\varphi_2, \sigma, t') \geq \epsilon \wedge \forall t'' \in [t, t'], \rho_\tau(\varphi_1, \sigma, t'') \geq \epsilon.$$

If $\forall t'' \in [t, t'], \rho_\tau(\varphi_1, \sigma, t'') \geq \epsilon$, then $\inf\limits_{t'' \in [t, t']} \rho_\tau(\varphi_1, \sigma, t'') \geq \epsilon$. In summary,

$$\exists t' \in (t + I) \cap [0, \tau), (\rho_\tau(\varphi_2, \sigma, t') \geq \epsilon \wedge \inf_{t'' \in [t, t']} \rho_\tau(\varphi_1, \sigma, t'') \geq \epsilon).$$

Since $\exists t' \in t + I \cap [0,\tau), \min(\rho_\tau(\varphi_2,\sigma,t'), \inf_{t'' \in [t,t']} \rho_\tau(\varphi_1,\sigma,t'')) \geq \epsilon$,

$$\sup_{t' \in t+I \cap [0,\tau)} \min(\rho_\tau(\varphi_2,\sigma,t'), \inf_{t'' \in [t,t']} \rho_\tau(\varphi_1,\sigma,t'')) \geq \epsilon.$$

Therefore, $\rho_\tau(\varphi,\sigma,t) \geq \epsilon$ by definition.

Suppose $\sigma, t \not\models_\tau (\varphi_1 \mathbf{U}_I \varphi_2)^{-\epsilon}$. Then, for all time points $t' \in (t+I) \cap [0,\tau)$, $\sigma, t' \not\models_\tau \varphi_2^{-\epsilon} \vee \exists t'' \in [t,t'], \sigma, t'' \not\models_\tau \varphi_1^{-\epsilon}$ by definition. Therefore,

$$\forall t' \in (t+I) \cap [0,\tau).(\rho_\tau(\varphi_2,\sigma,t') \leq -\epsilon \vee \exists t'' \in [t,t'], \rho_\tau(\varphi_1,\sigma,t'') \leq -\epsilon)$$

by induction hypothesis. If $\exists t'' \in [t,t'], \rho_\tau(\varphi_1,\sigma,t'') \leq -\epsilon$, then $\inf_{t'' \in [t,t']} \rho_\tau(\varphi_1,\sigma,t'') \leq -\epsilon$. In summary,

$$\forall t' \in (t+I) \cap [0,\tau), (\rho_\tau(\varphi_2,\sigma,t') \leq -\epsilon \vee \inf_{t'' \in [t,t']} \rho_\tau(\varphi_1,\sigma,t'') \leq -\epsilon).$$

Since $\forall t' \in t + I \cap [0,\tau).(\min(\rho_\tau(\varphi_2,\sigma,t'), \inf_{t'' \in [t,t']} \rho_\tau(\varphi_1,\sigma,t'')) \leq -\epsilon)$,

$$\sup_{t' \in t+I \cap [0,\tau)} \min(\rho_\tau(\varphi_2,\sigma,t'), \inf_{t'' \in [t,t']} \rho_\tau(\varphi_1,\sigma,t'')) \leq -\epsilon.$$

Therefore, $\rho_\tau(\varphi,\sigma,t) \leq -\epsilon$ by definition. □

According to above lemmas, we can reduce finding a counterexample for robust STL model checking of an STL formula φ into finding a counterexample for Boolean STL model checking of the ϵ-strengthening $\varphi^{+\epsilon}$.

Theorem 2. *For a hybrid automaton H, an STL formula φ, a time t, a time bound τ, and an arbitrary positive real value ϵ, the following properties are hold:*

(1) $\exists \sigma \in H.\ \sigma, t \models_\tau \neg(\varphi^{+\epsilon})$ implies $\exists \sigma \in H.\ \rho_\tau(\neg\varphi,\sigma,t) \geq -\epsilon$, and
(2) $\forall \sigma \in H.\ \sigma, t \not\models_\tau \neg(\varphi^{+\epsilon})$ implies $\forall \sigma \in H.\ \rho_\tau(\varphi,\sigma,t) \geq \epsilon$.

Proof. (1) Suppose there is a trajectory $\sigma \in H.\ \sigma, t \models_\tau \neg(\varphi^{+\epsilon})$. Then, the robustness degree $\rho_\tau(\varphi,\sigma,t) \leq \epsilon$ by Lemma 1. If the robustness degree $\rho_\tau(\varphi,\sigma,t)$ is less than or equal to ϵ, then $-\rho_\tau(\varphi,\sigma,t) \geq -\epsilon$. It is equivalent to $\rho_\tau(\neg\varphi,\sigma,t) \geq -\epsilon$, since $\rho_\tau(\neg\varphi,\sigma,t) = -\rho_\tau(\varphi,\sigma,t)$ by definition.
(2) Suppose all trajectories $\forall \sigma \in H.\ \sigma, t \not\models_\tau \neg(\varphi^{+\epsilon})$. The formula $\sigma, t \not\models_\tau \neg(\varphi^{+\epsilon})$ is equivalent to $\sigma, t \models_\tau \varphi^{+\epsilon}$ by definition. If a trajectory satisfies $\varphi^{+\epsilon}$ at time t, then the robustness degree $\rho_\tau(\varphi,\sigma,t) \geq \epsilon$ by Lemma 2. □

As a consequence, a counterexample of $\varphi^{+\epsilon}$ for Boolean STL model checking is also a counterexample for robust STL model checking of φ with respect to a robustness threshold ϵ. If there is no counterexample of $\varphi^{+\epsilon}$ for Boolean STL model checking, then it is guaranteed that the robustness degree of φ is always greater than or equal to ϵ. Therefore, for any positive robustness threshold $0 < \epsilon' < \epsilon$, φ is satisfied at t in H with respect to ϵ'. It is worth noting that φ may not be satisfied with respect to ϵ itself.

Algorithm 1: Two-Step SMT Solving Algorithm

```
Input: Hybrid automaton $H$, STL formula $\varphi$, threshold $\epsilon$, bounds $\tau$ and $N$
for $k = 1$ to $N$ do
    $\overline{\Psi} \leftarrow$ abstraction of the encoding $\Psi^{k,\tau}_{H,\neg(\varphi^{+\epsilon})}$ without flow and inv;
    while checkSat($\overline{\Psi}$) is Sat do
        $\pi \leftarrow$ a minimal satisfying scenario;
        $\hat{\pi} \leftarrow$ the refinement of $\pi$ with flow and inv;
        if checkSat($\hat{\pi}$) is Sat then
            return counterexample(result.satAssignment);
        $\overline{\Psi} \leftarrow \overline{\Psi} \wedge \neg\pi$;
return True;
```

4.2 Boolean STL Bounded Model Checking

For Boolean STL model checking, there exist *refutationally complete* bounded model checking algorithms [6, 34]. There are two bound parameters for these algorithms: a *time bound* τ that restricts a time domain of a hybrid automaton, and a *discret bound* N that restricts the number of mode changes and the number of variable points at which the truth value of some STL subformula changes. The algorithms build an SMT encoding $\Psi^{N,\tau}_{H,\neg\varphi}$ of Boolean STL model checking:

Theorem 3. *[6, 34] $\Psi^{N,\tau}_{H,\neg\varphi}$ is satisfiable iff there is a counterexample trajectory $\sigma \in H$, with at most N variable points and mode changes, such that $\sigma, t \not\models_\tau \varphi$.*

The satisfiability of $\Psi^{N,\tau}_{H,(\neg\varphi)}$ can be (approximately) determined using an SMT solver. When the flow conditions are linear or polynomial, the satisfiability can be precisely determined using Z3 [14] and Yices2 [19]. For hybrid automata with ODE dynamics, the satisfiability is undecidable in general, but a solver like dReal [26] can approximately determine its satisfiability.

4.3 Two-Step Solving Algorithm

The computation cost of dReal for ODEs is highly expensive. To reduce the complexity, we propose a two-step SMT solving algorithm, summariazed in Algorithm 1, inspired by the lazy SMT solving approach [43]:

1. We obtain the *discrete abstraction* of the encoding Ψ by substituting the flow and invariant conditions with Boolean variables. We then enumerate a satisfying *scenario* π, a conjunction of literals, where π implies Ψ.
2. For each scenario π, we check the satisfiability of its *discrete refinement* with the flow and invariant conditions using dReal. If any refinement is satisfiable, we obtain a counterexample; otherwise, there is no counterexample.

A naive implementation of the two-step solving algorithm may generate many redundant scenarios for hybrid automata.

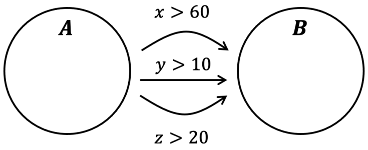

Fig. 8: Jump conditions between two modes

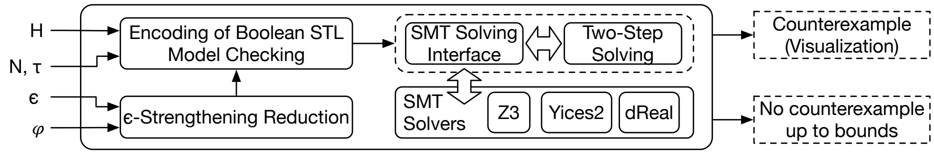
H
N, τ
ϵ
φ
Encoding of Boolean STL Model Checking
ϵ-Strengthening Reduction
SMT Solving Interface
Two-Step Solving
SMT Solvers
Z3
Yices2
dReal
Counterexample (Visualization)
No counterexample up to bounds

Fig. 9: The STLmc architecture

For example, in Figure 8, suppose there are two modes A and B, jump conditions from A to B are $x > 60$, $y > 10$, and $z > 20$. There are 7 possible scenarios from these conditions for a single jump. However, when one jump, say $x > 60$, is taken, the guards for the other jumps, i.e., $y > 10$ and $z > 20$, are not important. It suffices to consider only 3 scenarios, namely, $x > 60$, $y > 10$, and $z > 20$.

We implement a simple algorithm to minimize scenarios. A conjunction of literals $\pi = l_1 \wedge \cdots \wedge l_m$ is a minimal scenario, if $\pi' = l_1 \wedge \cdots l_{i-1} \wedge l_{i+1} \wedge \cdots l_m$ is not a scenario for any i. The conjunction π' is not a scenario when π' does not imply the encoding Ψ. It means $\neg(\pi' \Rightarrow \Psi)$ is satisfied. Thus, if $\pi' \wedge \neg\Psi$ is satisfied, π is the minimal scenario.

To minimize a scenario π, we apply a dual propagation approach [38]. Because π implies Ψ, the formula $\pi \wedge \neg\Psi$ is unsatisfiable. A conjunction of literals in the unsatisfiable core of $\pi \wedge \neg\Psi$ is a minimal scenario. We evaluate the unsatisfiable core of $\pi \wedge \neg\Psi$ using Z3. To obtain the unsatisfiable core, we encode the negation of the encoding Ψ in a separate dual solver under the assumptions of a set of literals in π using *assert_ and_ track* in Z3. To get a minimal unsatisfiable core, we use the option *smt.core.minimize true* for Z3.

4.4 Implementation

Figure 9 shows the overall architecture of STLmc. The given STL formula φ is first translated into the ϵ-strengthening of the negated formula $\neg(\varphi^{+\epsilon})$. The SMT encoding $\Psi^{k,\tau}_{H,\neg(\varphi^{+\epsilon})}$ for $1 \leq k \leq N$ is then built using the STL bounded model checking algorithm [6, 34]. The satisfiability of $\Psi^{k,\tau}_{H,\neg(\varphi^{+\epsilon})}$ can be checked directly using an SMT solver or using the two-step optimization algorithm. Our tool is implemented in around 9,500 lines of Python code.

Simplifying Universal Quantification. The encoding $\Psi^{k,\tau}_{H,(\neg\varphi)^{-\epsilon}}$ includes universal quantification over time. Such $\exists\forall$-conditions are supported by several solvers but involve high computational costs. STLmc implements the following methods to simplify universal quantification. For polynomial hybrid automata, we use the quantifier-free encoding [12] to encode ∃∀-conditions as quantifier-free formulas. For ODE dynamics, dReal natively supports ∃∀-conditions [27]. For invariant STL properties of the form $p \rightarrow \Box_I q$, we reduce redundant universal quantifiers by reducing the model checking problem into the reachability [34].

Parallelizing Two-step Solving. We parallelize the two-step solving algorithm for ODEs. As mentioned, the first phase generates abstract scenarios, and the second phase checks the feasibility of the scenarios. We simply run the feasibility checking of different scenarios in parallel. If any of such scenario is satisfied, then a counterexample is found and all other jobs are terminated. If all scenarios, checking in parallel, are unsatisfiable, then there is no counterexample. As shown in Sec 5, it greatly improves the performance for the ODE cases in practice.

Supporting Various SMT Solvers. We implement a generic wrapper interface based on the SMT-LIB standard [8] to support various SMT solvers. Therefore, it is easy to extend our tool with a new SMT solvers, if it follows SMT-LIB. Moreover, STLmc can also detect the most suitable solver for a given input model; e.g, if the model has ODE dynamics, then the tool chooses dReal. Given an input model, STLmc can detect the most suitable solvers. If the model has ODE dynamics and Z3 is selected, the tool raises an exception. Also, when the model has linear dynamics and the dReal is chosen, STLmc gives a warning to use Z3 or Yices instead of dReal.

5 Experimental Evaluation

We evaluate the effectiveness of the STLmc model checker using a number of hybrid system benchmarks and nontrivial STL properties. We use the following models, adapted from existing benchmarks [3,5,6,22–24,29,39]: load management for two batteries (Bat), two networked water tank systems (Wat), autonomous driving of two cars (Car), a railroad gate (Rail), two networked thermostats (Thm), docking of spacecraft (Space), navigation of a vehicle (Nav), and a filtered oscillator (Oscil). The models and the experimental results are available at https://stlmc.github.io/cav2022.

Section 5.1 shows the experiments on robust STL model checking. Section 5.2 compares the performance of STLmc for invariant properties with the existing reachability analysis tools for hybrid automata. We have run all experiments on Intel Xeon 2.8GHz with 256 GB memory.

5.1 STL Model Checking of Hybrid Automata

We measure the SMT encoding size and execution time for robust STL model checking. We consider three variants of continuous dynamics (linear, polynomial, and ODE dynamics) for the five models: Bat, Wat, Car, Rail, and Thm. We also consider the additional three models with ODE dynamics: Oscil, Space, and Nav. For each model, we use three STL formulas with nested temporal operators.

For discrete bounds, we use $N = 20$ for linear models, $N = 10$ for polynomial models, and $N = 5$ for ODE models. We use different time bounds τ and robustness thresholds ϵ for different models, since τ and ϵ depend on each model. As an underlying SMT solver, we use Yices for linear and polynomial models, and dReal for ODE models with a precision $\delta = 0.001$. We run both direct SMT solving (`1-step`) and two-step SMT solving (`2-step`). We use 25 cores for parallelizing the two-step SMT solving. We set a timeout of 1 hours.

The experimental results for linear and polynomial models are summarized in Table 2 and the experimental results for ODE models are summarized in Table 3, where τ denotes time bounds, and $|\Psi|$ denotes the size of the SMT encoding Ψ (in thousands) as the number of connectives in Ψ. For the model checking results, $\top$ indicates that the tool found no counterexample up to bound N, and $\bot$ indicates that the tool found a counterexample at bound $k \leq N$. For the algorithms (Alg.), we write one of the results with a better performance. For the `2-step` case, we also write the number of minimal scenarios generated ($\#\pi$).

As shown in Table 2 and 3, our tool can perform robust model checking of nontrivial STL formulas for hybrid systems with different continuous dynamics. The cases of ODE models generally take longer than the cases of linear and polynomial models, because of the high computational costs for ODE solving. Nevertheless, our parallelized two-step SMT solving method works well and all model checking analyses are finished before the timeout. In contrast, for linear and polynomial models with a larger discrete bound $N \geq 10$, direct SMT solving is usually effective but the two-step SMT solving method is not. There are too many scenarios, and the scenario generation does not terminate within 30 minutes. Therefore, the two algorithms implemented in our tool are complementary.

Table 2: Robust Bounded Model Checking of STL (Time in seconds)

Dyn.	Model	STL formula	τ	ϵ	$\lvert\Psi\rvert$	Time (s)	Result	k	Alg.
Linear (N = 20)	Bat	$\Diamond_{[4,10]}(p_1 \to \Box_{[4,10]}\, p_2)$	30	0.1	12.9	137	$\top$	-	1-step
		$(\Diamond_{[1,5]}\, p_1)\, \mathbf{R}_{[5,20]}\, p_2$		3.5	2.76	5.71	$\bot$	5	1-step
		$\Box_{[4,14]}(p_1 \to \Diamond_{[0,10]}\, p_2)$		0.1	3.8	22.1	$\bot$	8	1-step
	Wat	$\Box_{[1,3]}(p_1\, \mathbf{R}_{[1,10]}\, p_2)$	20	2.5	18.8	26.2	$\top$	-	1-step
		$(\Diamond_{[1,10]}\, p_1)\, \mathbf{U}_{[2,5]}\, p_2$		0.1	1.9	4.22	$\bot$	4	1-step
		$\Diamond_{[4,10]}(p_1 \to \Box_{[2,5]}\, p_2)$		0.01	11.2	20.2	$\top$	-	1-step
	Car	$(\Diamond_{[3,5]}\, p_1)\, \mathbf{U}_{[2,10]}\, p_2$	40	0.1	2.5	12.1	$\bot$	5	1-step
		$\Diamond_{[3,30]}(\Box_{[5,7]}\, p_1)$		2	2.9	9.96	$\bot$	8	1-step
		$(\Box_{[2,5]}\, p_1)\, \mathbf{R}_{[0,10)}\, p_2$		1	2.5	11.8	$\bot$	5	1-step
	Rail	$\Diamond_{[3,10]}(p_1\, \mathbf{U}_{[0,10]}\, p_2)$	30	2	17.9	25.3	$\top$	-	1-step
		$(\Box_{[2,10]}\, p_1)\, \mathbf{R}_{[0,15]}\, p_2$		1	1.25	5.58	$\bot$	3	1-step
		$\Box_{[0,5]}(\Diamond_{[0,10]}\, p_1)$		0.5	0.8	4.16	$\bot$	3	1-step
	Thm	$\Diamond_{[2,5]}(p_1\, \mathbf{R}_{[2,10]}\, p_2)$	30	3	16.8	24.5	$\top$	-	1-step
		$\Box_{[1,4]}(p_1 \to \Diamond_{[5,15]}\, p_2)$		1	11.1	18	$\top$		1-step
		$p_1\, \mathbf{U}_{[10,20]}(\Box_{[3,5]}\, p_2)$		2.5	0.98	5.52	$\bot$	2	1-step
Poly (N = 10)	Bat	$\Box_{[3,5]}(p_1 \to \Diamond_{[6,10]}\, p_2)$	25	2.5	1.7	4.75	$\bot$	4	1-step
		$\Diamond_{[4,14]}(p_1\, \mathbf{U}_{[4,10]}\, p_2)$		1	2.3	5.51	$\bot$	4	1-step
		$\Diamond_{[0,15]}(p_1\, \mathbf{U}_{[0,6]}\, p_2)$		0.5	7.2	59.9	$\top$	-	1-step
	Wat	$\Box_{[1,4]}(p_1\, \mathbf{U}_{[2,5]}\, p_2)$	10	1.5	0.96	2.75	$\bot$	2	1-step
		$\Diamond_{[0,3]}(\Box_{[0,4]}\, p_1)$		2	0.95	2.3	$\bot$	3	1-step
		$\Diamond_{[2,4]}(p_1 \to \Box_{[3,5]}\, p_2)$		4	0.68	3.1	$\bot$	2	1-step
	Car	$\Box_{[0,4]}(p_1 \to \Diamond_{[2,5]}\, p_2)$	10	0.5	2.2	7.24	$\bot$	5	1-step
		$(\Diamond_{[0,4]}\, p_1)\, \mathbf{U}_{[0,5]}\, p_2$		2.0	1.7	6.27	$\bot$	3	1-step
		$\Diamond_{[0,3]}(p_1\, \mathbf{U}_{[0,5]}\, p_2)$		0.1	7.3	9.72	$\top$	-	1-step
	Rail	$\Diamond_{[0,5]}(p_1\, \mathbf{U}_{[1,8]}\, p_2)$	20	1.0	2.3	3.43	$\bot$	5	1-step
		$\Diamond_{[0,4]}(p_1 \to \Box_{[2,10]}\, p_2)$		5.0	3.8	0.86	$\top$	-	1-step
		$(\Box_{[0,5)}\, p_1)\, \mathbf{U}_{[2,10]}\, p_2$		4.0	1.9	2.83	$\bot$	4	1-step
	Thm	$(\Box_{[2,10]}\, p_1)\, \mathbf{U}_{[1,4]}\, p_2$	10	0.5	2.0	2.7	$\bot$	4	1-step
		$\Diamond_{[0,5]}(p_1 \to \Box_{[2,5)}\, p_2)$		0.1	3.9	5.4	$\top$	-	1-step
		$\Diamond_{[0,10]}(p_1\, \mathbf{R}_{[2,4]}\, p_2)$		1	5.9	6.8	$\top$	-	1-step

5.2 Comparison with Reachability Analysis Tools

We compare the performance of STLmc for invariant properties with four reachability analysis tools for hybrid automata: HyComp [11], SpaceEx [24], Flow* [10], and dReach [33]. For each model, we consider a true invariant property so that reachability analysis explores all possible behaviors up to given bounds. We measure the execution times (in seconds) for analyzing the invariant properties with a timeout of 1 hours.

Since each tool has a different notion of bounds, we use the number of discrete jumps N and the maximum time horizon τ as the common bound parameters

Table 3: Robust Bounded Model Checking of STL (N = 5 and time in seconds)

Model	STL formula	τ	ϵ	$\lvert\Psi\rvert$	Time (s)	Result	k	Alg.	$\#\pi$
Bat	$\Diamond_{[0,4]}(p_1\,\mathbf{R}_{[0,5]}\,p_2)$	10	1	1.3	15.6	$\bot$	3	`2-step`	60
	$\Box_{[3,4]}(p_1 \to \Diamond_{[0,6]}\,p_2)$		1.5	0.9	37.8	$\bot$	3	`2-step`	186
	$\Box_{[1,3]}(\Diamond_{[0,5]}\,p_1)$		1	0.8	107	$\bot$	3	`2-step`	482
Thm	$\Diamond_{[0,3]}(p_1\,\mathbf{U}_{[0,\infty)}\,p_2)$	30	1.0	1.2	817	$\top$	-	`2-step`	3,646
	$\Box_{[2,4]}(p_1 \to \Diamond_{[3,10]}\,p_2)$		2.0	0.7	7.46	$\bot$	2	`2-step`	47
	$\Box_{[0,10]}(p_1\,\mathbf{R}_{[0,\infty)}\,p_2)$		2.0	1.2	59.3	$\bot$	4	`2-step`	212
Oscil	$\Diamond_{[0,3]}(p_1\,\mathbf{R}_{[0,\infty)}\,p_2)$	8	0.1	1.5	110	$\top$	-	`2-step`	289
	$\Diamond_{[2,5]}(\Box_{[0,3]}\,p_1)$		1.0	1.2	224	$\bot$	3	`2-step`	259
	$(\Box_{[1,3]}\,p_1)\,\mathbf{R}_{[2,5]}\,p_2$		0.1	1.2	266	$\bot$	3	`2-step`	266
Space	$\Box_{[0,2]}(p_1 \to \Diamond_{[0,3]}\,p_2)$	5	1.5	0.8	1692	$\bot$	2	`2-step`	116
	$\Diamond_{[2,3]}(\Box_{[1,2]}\,p_1)$		0.1	1.1	69.1	$\bot$	3	`2-step`	224
	$\Box_{[0,2]}(p_1 \to \Diamond_{[1,3]}\,p_2)$		1	0.8	10.6	$\bot$	2	`2-step`	42
Nav	$\Diamond_{[2,4]}(p_1 \to \Box_{[1,5]}\,p_2)$	10	3	1.2	358	$\bot$	3	`2-step`	1197
	$\Diamond_{[2,4]}(\Box_{[3,6]}\,p_1)$		2	1.1	285	$\bot$	3	`2-step`	1000
	$\Diamond_{[1,5]}(p_1\,\mathbf{R}\,p_2)$		1	1.4	949	$\top$	-	`2-step`	2432
Wat	$p_1\,\mathbf{R}_{[0,3]}(\Diamond_{[2,4]}\,p_2)$	8	2	1.1	32.8	$\bot$	3	`2-step`	140
	$\Box_{[1,3]}(p_1 \to \Diamond_{[0,5]}\,p_2)$		0.1	1.3	2549	$\top$	-	`2-step`	6492
	$\Box_{[2,4]}(\Diamond_{[0,4]}\,p_1)$		0.5	0.7	14.5	$\bot$	3	`2-step`	95
Car	$\Diamond_{[1,3]}(\Box_{[0,2]}\,p_1)$	5	1	1.2	25.9	$\bot$	2	`2-step`	90
	$(\Diamond_{[1,2]}\,p_1)\,\mathbf{R}_{[0,2]}\,p_2$		0.5	1.5	69.4	$\bot$	2	`2-step`	114
	$\Box_{[0,2]}(p_1 \to \Diamond_{[2,3]}\,p_2)$		2	1.2	23.2	$\bot$	2	`2-step`	92
Rail	$\Box_{[1,3]}(p_1 \to \Diamond_{[2,4]}\,p_2)$	8	2	0.8	10.9	$\bot$	3	`2-step`	62
	$(\Diamond_{[1,3]}\,p_1)\,\mathbf{U}_{[1,4]}\,p_2$		0.5	1.3	80.3	$\bot$	3	`2-step`	228
	$\Diamond_{[0,4]}(\Box_{[2,4]}\,p_1)$		1	0.8	121	$\bot$	3	`2-step`	55

(which are "encoded" in the models) if needed. For STLmc, we use direct SMT solving (`1-step`) and Yices as an underlying SMT solver for linear and polynomial models. For ODE models, we use two-step SMT solving (`2-step`) and dReal. For SpaceEx, we use PHAVer for linear models, and STC for polynomial and ODE models. For Flow*, we use adaptive steps and TM orders 1 (for linear), 2 (for polynomial) and $[2, 15]$ (for ODE). For dReach, we set precision to 0.1. We use BMC for HyComp.

The experimental results are summarized in Table 4, with execution times in seconds. `T/O` denotes time out. HyComp only supports linear models, and so the results for polynomial and ODE models do not include HyComp. SpaceEx only supports linear ODE, and so is not applicable (`N/A`) for models with nonlinear ODEs (Car, Rail, and Wat). As seen, STLmc has comparable performance to the other tools for the invariant properties.

Table 4: Execution time for analyzing invariant properties (in seconds)

Model	Linear ($N = 20$)					Poly ($N = 10$)			
	STLmc	HyComp	SpaceEx	Flow*	dReach	STLmc	SpaceEx	Flow*	dReach
Bat	0.25	7.54	0.07	44	3.54	0.44	0.01	19.4	0.58
Wat	0.18	1.56	0.49	128	T/O	2.51	0.001	0.08	2116
Car	0.21	0.74	1.0	26.5	T/O	0.32	2607	27	41.7
Rail	0.14	0.77	0.14	3.91	1158	0.07	0.25	2.73	0.09
Thm	0.21	1.25	0.59	402	T/O	0.34	3.25	5.21	T/O

Table 5: Execution time (in seconds) for analyzing invariant properties (N = 5)

Model	STLmc	SpaceEx	Flow*	dReach	Model	STLmc	SpaceEx	Flow*	dReach
Bat	0.1	0.03	4.53	0.1	Nav	0.1	0.02	1.0	0.07
Thm	1.3	0.24	16.4	2.3	Wat	0.06	N/A	167	0.04
Oscil	0.07	0.02	0.48	0.17	Car	39.2		T/O	47.8
Space	0.06	0.02	0.42	2.6	Rail	0.1		0.08	0.08

6 Related Work

There exist many tools for falsifying STL properties of hybrid systems, including Breach [16], S-talrio [4], and TLTk [13]. STL falsification techniques are based on STL monitoring [15, 17, 30, 37], and often use stochastic optimization techniques, such as Ant-Colony Optimization [4], Monte-Carlo tree search [47], deep reinforcement learning [2], and so on. These techniques are often quite useful for finding counterexamples in practice, but, as mentioned, cannot be used to verify STL properties of hybrid systems.

There exist many tools for analyzing reachability properties of hybrid systems based on reachable-set computation, including C2E2 [18], Flow* [10], Hylaa [7], and SpaceEx [24]. They can be used to guarantee the correctness of invariant properties of the form $p \to \Box_I\, q$, but cannot verify general STL properties. In contrast, STLmc uses a refutation-complete bounded STL model checking algorithm to verify general STL properties, including complex ones.

Our tool is also related to SMT-based tools for analyzing hybrid systems, including dReach [33], HyComp [11], and HybridSAL [44]. These techniques also focus on analyzing invariant properties of hybrid systems, but some SMT-based tools, such as HyComp, can verify LTL properties of hybrid systems. Unlike STLmc, they cannot deal with general STL properties of hybrid systems.

7 Concluding Remarks

We have presented the STLmc tool for robust bounded model checking of STL properties for hybrid systems. STLmc can verify that, up to given bounds, the robustness degree of an STL formula φ is always greater than a given robustness threshold for all possible behaviors of a hybrid system. STLmc also provides a convenient user interface with an intuitive counterexample visualization.

Our tool leverages the reduction from robust model checking to Boolean model checking, and utilizes the refutation-complete SMT-based Boolean STL model checking algorithm to guarantee correctness up to given bounds and find subtle counterexamples. STLmc can deal with hybrid systems with (nonlinear) ODEs using dReal. We have shown using various hybrid system benchmarks that STLmc can effectively analyze nontrivial STL properties.

Future work includes extending our tool with other hybrid system analysis methods, such as reachable-set computation, besides SMT-based approaches.

References

1. Abbas, H., Fainekos, G., Sankaranarayanan, S., Ivančić, F., Gupta, A.: Probabilistic temporal logic falsification of cyber-physical systems. ACM Transactions on Embedded Computing Systems (TECS) **12**(2s), 1–30 (2013)
2. Akazaki, T., Liu, S., Yamagata, Y., Duan, Y., Hao, J.: Falsification of cyber-physical systems using deep reinforcement learning. In: International Symposium on Formal Methods. pp. 456–465. Springer (2018)
3. Alur, R.: Principles of cyber-physical systems. The MIT Press (2015)
4. Annpureddy, Y., Liu, C., Fainekos, G., Sankaranarayanan, S.: S-taliro: A tool for temporal logic falsification for hybrid systems. In: Proc. TACAS. LNCS, vol. 6605, pp. 254–257. Springer (2011)
5. Bae, K., Gao, S.: Modular smt-based analysis of nonlinear hybrid systems. In: Proceedings of the 17th Conference on Formal Methods in Computer-Aided Design. pp. 180–187. FMCAD '17, FMCAD, Austin, TX (2017), http://dl.acm.org/citation.cfm?id=3168451.3168490
6. Bae, K., Lee, J.: Bounded model checking of signal temporal logic properties using syntactic separation. Proc. ACM Program. Lang. **3, POPL**(51), 1–30 (2019)
7. Bak, S., Duggirala, P.S.: Hylaa: A tool for computing simulation-equivalent reachability for linear systems. In: Proc. HSCC. pp. 173–178. ACM (2017)
8. Barrett, C., Stump, A., Tinelli, C., et al.: The SMT-LIB standard: Version 2.0. In: Proceedings of the 8th international workshop on satisfiability modulo theories (Edinburgh, England). vol. 13, p. 14 (2010)
9. Chen, G., Liu, M., Kong, Z.: Temporal-logic-based semantic fault diagnosis with time-series data from industrial internet of things. IEEE Transactions on Industrial Electronics **68**(5), 4393–4403 (2020)
10. Chen, X., Ábrahám, E., Sankaranarayanan, S.: Flow*: An analyzer for non-linear hybrid systems. In: Proc. CAV. LNCS, vol. 8044, pp. 258–263. Springer (2013)
11. Cimatti, A., Griggio, A., Mover, S., Tonetta, S.: HyComp: an SMT-based model checker for hybrid systems. In: Proc. TACAS. LNCS, vol. 9035. Springer (2015)
12. Cimatti, A., Mover, S., Tonetta, S.: A quantifier-free smt encoding of non-linear hybrid automata. In: Formal Methods in Computer-Aided Design (FMCAD), 2012. pp. 187–195. IEEE (2012)

13. Cralley, J., Spantidi, O., Hoxha, B., Fainekos, G.: Tltk: A toolbox for parallel robustness computation of temporal logic specifications. In: International Conference on Runtime Verification. pp. 404–416. Springer (2020)
14. De Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Proc. TACAS. LNCS, vol. 4963, pp. 337–340. Springer (2008)
15. Deshmukh, J.V., Donzé, A., Ghosh, S., Jin, X., Juniwal, G., Seshia, S.A.: Robust online monitoring of signal temporal logic. Form. Methods Syst. Des. **51**(1) (2017)
16. Donzé, A.: Breach, a toolbox for verification and parameter synthesis of hybrid systems. In: Proceedings of the 22nd International Conference on Computer Aided Verification. pp. 167–170. CAV'10, Springer (2010)
17. Donzé, A., Ferrère, T., Maler, O.: Efficient robust monitoring for STL. In: Proc. CAV. LNCS, vol. 8044. Springer (2013)
18. Duggirala, P.S., Mitra, S., Viswanathan, M., Potok, M.: C2e2: A verification tool for stateflow models. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 68–82. Springer (2015)
19. Dutertre, B.: Yices 2.2. In: Biere, A., Bloem, R. (eds.) Proc. CAV. LNCS, vol. 8559, pp. 737–744. Springer (2014)
20. Eddeland, J.L., Donzé, A., Miremadi, S., Åkesson, K.: Industrial temporal logic specifications for falsification of cyber-physical systems. ARCH (2020)
21. Farahani, S.S., Soudjani, S.E.Z., Majumdar, R., Ocampo-Martinez, C.: Robust model predictive control with signal temporal logic constraints for barcelona wastewater system. IFAC-PapersOnLine **50**(1), 6594–6600 (2017)
22. Fehnker, A., Ivančić, F.: Benchmarks for hybrid systems verification. In: Alur, R., Pappas, G.J. (eds.) Hybrid Systems: Computation and Control. pp. 326–341. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
23. Fox, M., Long, D., Magazzeni, D.: Plan-based policies for efficient multiple battery load management. JAIR **44**, 335–382 (2012)
24. Frehse, G., Le Guernic, C., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., Maler, O.: SpaceEx: Scalable verification of hybrid systems. In: Proc. CAV. LNCS, vol. 6806, pp. 379–395. Springer (2011)
25. Gao, S., Avigad, J., Clarke, E.M.: Delta-decidability over the reals. In: 2012 27th Annual IEEE Symposium on Logic in Computer Science. pp. 305–314. IEEE (2012)
26. Gao, S., Kong, S., Clarke, E.M.: dReal: An SMT solver for nonlinear theories over the reals. In: Proc. CADE. LNCS, vol. 7898, pp. 208–214. Springer (2013)
27. Gao, S., Kong, S., Clarke, E.M.: Satisfiability modulo ODEs. In: Proc. FMCAD. pp. 105–112. IEEE (2013)
28. Goldman, R.P., Bryce, D., Pelican, M.J., Musliner, D.J., Bae, K.: A hybrid architecture for correct-by-construction hybrid planning and control. In: Proc. NFM. LNCS, vol. 9690. Springer (2016)
29. Henzinger, T.: The theory of hybrid automata. In: Verification of Digital and Hybrid Systems, NATO ASI Series, vol. 170, pp. 265–292. Springer (2000)
30. Jakšić, S., Bartocci, E., Grosu, R., Ničković, D.: Quantitative monitoring of STL with edit distance. Form. Methods Syst. Des. **53**(1), 83–112 (2018)
31. Jin, X., Deshmukh, J.V., Kapinski, J., Ueda, K., Butts, K.: Powertrain control verification benchmark. In: Proc. HSCC. ACM (2014)
32. Kapinski, J., Deshmukh, J.V., Jin, X., Ito, H., Butts, K.: Simulation-based approaches for verification of embedded control systems: An overview of traditional and advanced modeling, testing, and verification techniques. IEEE Control Systems Magazine **36**(6), 45–64 (2016)
33. Kong, S., Gao, S., Chen, W., Clarke, E.M.: dReach: δ-reachability analysis for hybrid systems. In: Proc. TACAS. LNCS, vol. 7898, pp. 200–205. Springer (2015)

34. Lee, J., Yu, G., Bae, K.: Efficient smt-based model checking for signal temporal logic. In: 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE). pp. 343–354 (2021). https://doi.org/10.1109/ASE51524.2021.9678719
35. Ma, M., Bartocci, E., Lifland, E., Stankovic, J., Feng, L.: Sastl: spatial aggregation signal temporal logic for runtime monitoring in smart cities. In: 2020 ACM/IEEE 11th International Conference on Cyber-Physical Systems (ICCPS). pp. 51–62. IEEE (2020)
36. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Proc. FORMATS. LNCS, vol. 3253, pp. 152–166. Springer (2004)
37. Ničković, D., Lebeltel, O., Maler, O., Ferrère, T., Ulus, D.: Amt 2.0: qualitative and quantitative trace analysis with extended signal temporal logic. In: Proc. TACAS. vol. 10806, pp. 303–319. Springer (2018)
38. Niemetz, A., Preiner, M., Biere, A.: Turbo-charging lemmas on demand with don't care reasoning. In: 2014 Formal Methods in Computer-Aided Design (FMCAD). pp. 179–186. IEEE (2014)
39. Raisch, J., Klein, E., Meder, C., Itigin, A., O'Young, S.: Approximating automata and discrete control for continuous systems — two examples from process control. In: Hybrid systems V. LNCS, vol. 1567, pp. 279–303. Springer (1999)
40. Roehm, H., Oehlerking, J., Heinz, T., Althoff, M.: STL model checking of continuous and hybrid systems. In: Proc. ATVA. LNCS, vol. 9938. Springer (2016)
41. Roohi, N., Kaur, R., Weimer, J., Sokolsky, O., Lee, I.: Parameter invariant monitoring for signal temporal logic. In: Proc. HSCC. pp. 187–196. ACM (2018)
42. Sankaranarayanan, S., Fainekos, G.: Falsification of temporal properties of hybrid systems using the cross-entropy method. In: Proceedings of ACM international conference on Hybrid Systems: Computation and Control. pp. 125–134 (2012)
43. Sebastiani, R.: Lazy satisfiability modulo theories. Journal on Satisfiability, Boolean Modeling and Computation **3**(3-4), 141–224 (2007)
44. Tiwari, A.: HybridSAL relational abstracter. In: Proc. CAV. Lecture Notes in Computer Science, vol. 7358, pp. 725–731. Springer (2012)
45. Wu, T., Olama, M.M., Djouadi, S.M., Dong, J., Xue, Y., Kuruganti, T.: Signal temporal logic control for residential hvac systems to accommodate high solar pv penetration. In: 2020 IEEE Power & Energy Society Innovative Smart Grid Technologies Conference (ISGT). pp. 1–5. IEEE (2020)
46. Xu, Z., Belta, C., Julius, A.: Temporal logic inference with prior information: An application to robot arm movements. IFAC-PapersOnLine **48**(27), 141–146 (2015)
47. Zhang, Z., Ernst, G., Sedwards, S., Arcaini, P., Hasuo, I.: Two-layered falsification of hybrid systems guided by Monte-Carlo tree search. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **37**(11) (2018)
48. Zhang, Z., Lyu, D., Arcaini, P., Ma, L., Hasuo, I., Zhao, J.: Effective hybrid system falsification using monte carlo tree search guided by qb-robustness. In: Proc. CAV. LNCS, vol. 12759, pp. 595–618. Springer (2021)